Indicateurs de performance d'un calcul (temps/mémoire)

1 But

Lors d'une simulation avec *Code_Aster*, **des affichages par défaut tracent** dans les fichiers message et résultats (`.mess/.resu`) **certaines caractéristiques dimensionnantes du calcul** (consommations RAM, temps CPU, système et utilisateur...). Cette documentation détaille les affichages de ces indicateurs de performance.

On formule aussi **quelques conseils pour aider l'utilisateur à tirer parti de ces diagnostics**. Mais il faut bien être conscient qu'il n'y a pas de recette universelle pour optimiser les performances globales d'un calcul. Cela dépend du type d'étude, des aspects logiciels et matériels de la machine, voire de sa charge !

Le paramétrage par défaut et les affichages/alarmes du code proposent un fonctionnement équilibré et instrumenté. Mais, pour être sûr d'avoir utilisé au mieux les capacités de sa machine, l'utilisateur doit rester attentif aux éléments décrits dans ce document ainsi qu'aux conseils présents dans les documentations des commandes. Ce document reprend de nombreux éléments des documentations Utilisateur sur le mot-clé `SOLVEUR` [U4.50.01], [U2.08.06] et sur le parallélisme [U2.08.03].

Table des Matières

2 Généralités

Lors d'une simulation *Code_Aster*, des affichages par défaut tracent dans les fichiers message et résultats (.mess/.resu) **certaines caractéristiques dimensionnantes du calcul**. On retrouve notamment, pour chaque opérateur *Aster* :

- Les *caractéristiques des systèmes linéaires* à construire et à résoudre (nombre de nœuds, d'équations et de Lagranges, taille de la matrice...),
- Les *mémoires JEVEUX* plancher (pour passer en Out-Of-Core[1]) et optimale (pour passer en In-Core),
- La *mémoire hors JEVEUX* requise par certains *produits externes* (par ex. MUMPS),
- Les *temps* CPU, système et «utilisateur» (elapsed),
- La *ventilation des temps* consommés suivant les étapes du calcul (calcul élémentaire, assemblage, résolution du système linéaire).

Cette dernière description des temps consommés peut se décliner suivant différents niveaux de lecture (impression synthétique, détaillée et détaillée par incrément de calcul) *via* le paramètre MESURE_TEMPS/NIVE_DETAIL des commandes DEBUT/POURSUITE. En mode parallèle, on rajoute la valeur moyenne, sur tous les processeurs, des temps consommés ainsi que leur écart-type.

1 L'Out-Of-Core (OOC) est un mode de gestion de la mémoire qui consiste à décharger sur disque certains objets alloués par le code pour libérer de la RAM. Ces déchargements peuvent être automatiques (déclenchés par le système ou le progiciel JEVEUX) ou organisés par le programmeur. La stratégie OOC permet de traiter des problèmes plus gros mais ces accès disque ralentissent le calcul. *A contrario*, le mode In-Core (IC) consiste à garder les objets en RAM. Cela limite la taille des problèmes accessibles, mais privilégie la vitesse.

3 Caractéristiques du système linéaire

Un grand nombre d'opérateurs (STAT/DYNA/THER_NON_LINE, CALC_MODES, ...) requièrent la construction et la résolution de systèmes linéaires. Pour résoudre ces systèmes d'équations, on utilise des algorithmes particuliers appelés «**solveurs linéaires**». Ces solveurs linéaires sont en fait omniprésents dans le déroulement des opérateurs de *Code_Aster* car ils sont souvent employés par les algorithmes numériques : schéma non linéaire, intégration en temps, analyse modale etc. Ils en consomment la majeure partie du temps CPU et de la mémoire.
Par défaut (INFO=1), chaque commande trace, lors de la construction de son premier système linaire, ses caractéristiques : taille N, nombre de termes non nuls NNZ ...

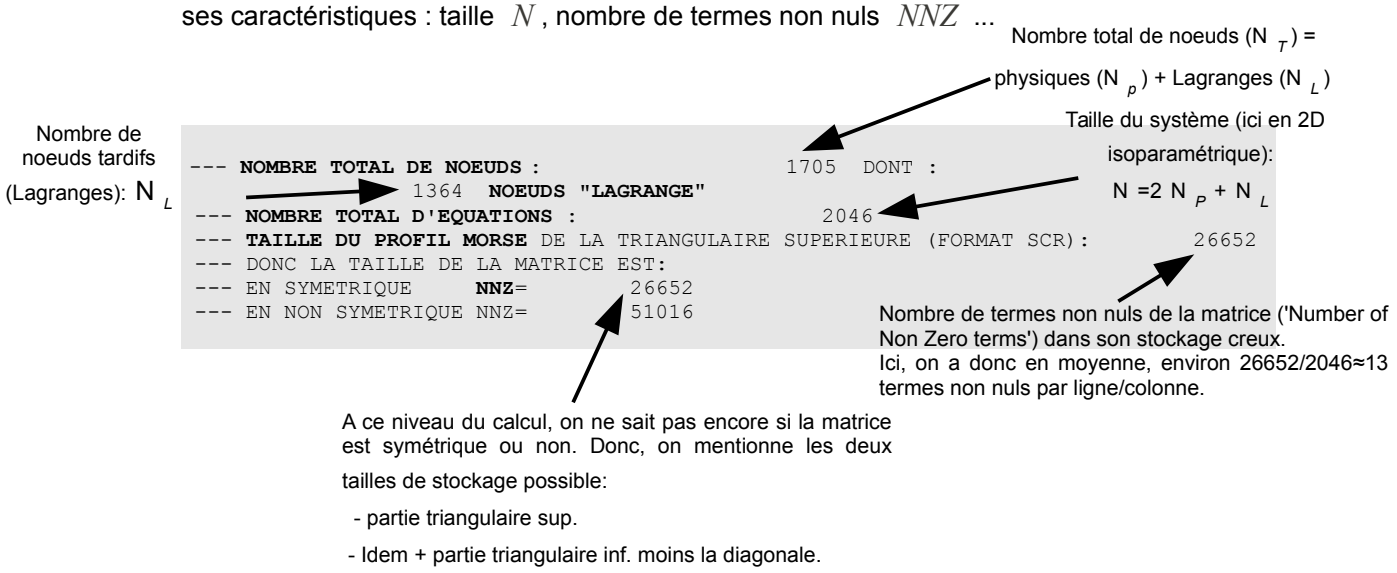

Figure 3-a : Caractéristiques du système linéaire en cours d'assemblage dans une commande Aster type STAT_NON_LINE ou THER_LINEAIRE (extrait du .mess).

La **taille du système**, la **proportion de Lagranges** et le **remplissage de la matrice** renseignent indirectement sur les **consommations mémoires/CPU** du calcul et sur d'éventuelles difficultés de résolutions.
A grands traits, la complexité algorithmique de la construction du système linéaire est en NNZ tandis que sa résolution en $N^{\alpha}.NNZ$ (avec $1<\alpha<2$).
L'occupation mémoire totale est elle du type[2] $\beta.NNZ$ (avec $10<\beta<100$) $\times 8\,octets$. Cette consommation mémoire regroupe plusieurs structures de données *Aster* et/ou liées à des produits externes (MATR_ASSE, NUME_DDL, matrice factorisée...). Heureusement, ces structures de données se décomposent en plusieurs segments mémoires distincts et l'algorithmique n'impose pas leur présence simultanée en mémoire RAM. On peut ainsi souvent décharger sur disque une bonne partie de ces données.
D'autre part, l'introduction de variables de type Lagranges[3] dans les matrices (matrices dites alors dualisées) dégrade leurs propriétés numériques (taille, définie-positivité, conditionnement matriciel). Cela implique donc souvent plus de traitements numériques dans les solveurs et dégrade leurs performances. Cette remarque peut d'ailleurs s'étendre à tous les éléments finis mixtes (modélisation incompressible, méthode continue en contact...).

2 β est grossièrement le facteur de remplissage de la factorisée. C'est-à-dire, le surfacteur en taille mémoire par rapport à la matrice initiale, qu'implique le processus de remplissage de la factorisation (factorisation totale en simple ou double précision de LDLT/MULT_FRONT/MUMPS ou factorisation simple précision des préconditionneurs GCPC/PETSC, [U4.50.01] et documentations de référence associées).
3 Ces variables de Lagrange sont introduites «artificiellement» en cours de calcul afin de prendre en compte les conditions de Dirichlet (simple/blocage ou généralisée/liaison).

Remarques:

L'affichage des caractéristiques du problème (figure 3.1) n'est effectué qu'une seule fois, en début de commande. En général, ces caractéristiques ne changent pas en cours de calcul. Les seules exceptions concernent l'opérateur `STAT_NON_LINE` *avec la méthode X-FEM en grand glissement ou la méthode continue en contact. Pour celles-ci, le profil de la matrice change au fur et à mesure des itérations.*

Cet affichage s'effectue très tôt dans le processus, à l'issue de la création du profil de la matrice[4] Aster. L'assemblage des termes élémentaires n'est pas encore effectué[5]. C'est pour cette raison qu'on ne peut statuer, a priori, sur le caractère symétrique ou non de la matrice. En pratique, elle est très souvent symétrique et, lorsque ce n'est pas le cas, on peut la symétriser via l'option `SOLVEUR/SYME`.

4 Procédure qui analyse les inconnues du problème (pour faire le lien noeud/ddl/inconnue algébrique) et dimensionne certaines structures de données Aster (NUME_DDL, MATR_ASSE..).

5 Il suffit qu'un terme élémentaire soit non symétrique pour que la matrice assemblée le devienne.

4 Ventilation des temps consommés

4.1 Temps consommés par commande

Code_Aster propose deux niveaux de lecture pour analyser les temps consommés par chaque commande :

- Un **niveau global** qui amalgame toutes les étapes de calcul de ladite commande,
- Un **niveau plus fin** (paramétrable) qui permet de dissocier les principales étapes.

4.1.1 Monitoring global de chaque commande

À l'issue de chaque commande, sont tracés dans le fichier message les différentes consommations en temps[6] de l'opérateur : CPU[7]+système (USER+SYST), système (SYST) et elapsed (ELAPS). ***A priori*, toute dérive importante du temps SYST et/ou du temps ELAPSED doit questionner (cf. §6.2).**

```
#  FIN COMMANDE NO : 0045   USER+SYST:  520.01s (SYST:  12.17s, ELAPS:    525.12s)
#  --------------------------------------------------------------------------------
```

Figure 4.1.1-a : Traces des consommations globales en temps d'une commande Aster (extrait d'un `.mess`).

Ces temps sont aussi récapitulés, pour toutes les commandes, dans le fichier résultat.

```
********************************************************************************
* COMMAND                 :       USER :     SYSTEM :   USER+SYS :    ELAPSED *
********************************************************************************
* init (jdc)              :       2.62 :       0.88 :       3.50 :       4.30 *
*  . compile              :       0.00 :       0.00 :       0.00 :       0.01 *
*  . exec_compile         :       0.54 :       0.03 :       0.57 :       0.58 *
*  . report               :       0.03 :       0.00 :       0.03 :       0.03 *
*  . build                :       0.00 :       0.00 :       0.00 :       0.00 *
* DEBUT                   :       0.04 :       0.05 :       0.09 :       0.13 *
* PRE_GIBI                :      10.75 :       1.89 :      12.64 :      12.69 *
* LIRE_MAILLAGE           :      27.92 :       0.13 :      28.05 :      28.37 *
* DEFI_MATERIAU           :       0.01 :       0.00 :       0.01 :       0.01 *
* AFFE_MATERIAU           :       0.04 :       0.01 :       0.05 :       0.05 *
* AFFE_MODELE             :       5.48 :       0.08 :       5.56 :       5.57 *
* AFFE_CHAR_MECA          :       0.52 :       0.02 :       0.54 :       0.54 *
                                    ...
* MECA_STATIQUE           :    2249.89 :      18.55 :    2268.44 :    2271.87 *
* TEST_RESU               :       0.01 :       0.01 :       0.02 :       0.01 *
* FIN                     :       0.11 :       0.01 :       0.12 :       0.17 *
*  . part Superviseur     :       3.37 :       0.94 :       4.31 :       5.21 *
*    . sdveri             :       0.68 :       0.01 :       0.69 :       0.73 *
*  . part Fortran         :    4600.81 :      39.42 :    4640.23 :    4650.96 *
********************************************************************************
* TOTAL_JOB               :    4604.18 :      40.36 :    4644.54 :    4656.18 *
********************************************************************************
```

Figure 4.1.1-b : Consommations globales en temps de toutes les commandes Aster (extrait d'un `.resu`).

4.1.2 Monitoring fin de chaque commande

La **ventilation des temps consommés** (USER+SYST, SYST, ELAPS) suivant les différentes étapes de calcul (calcul élémentaire, assemblage, factorisation numérique...) est effectuée à l'issue de chaque opérateur impliquant la construction et/ou la résolution de systèmes linéaires (par exemple STAT_NON_LINE et CALC_CHAMP). Par défaut, on affiche des valeurs synthétiques (niv=1). En effet, cette description des temps consommés peut se décliner suivant différents niveaux de lecture *via* le paramètre MESURE_TEMPS/NIVE_DETAIL des commandes DEBUT/POURSUITE.

6 Le temps CPU mesure l'exécution des sources du code (C, fortran, python). Le temps système prend en compte les appels systèmes sous-jacents (accès disque/RAM...). Le temps elapsed englobe les deux précédents et mesure le temps réel écoulé (« wall clock »).

7 Le temps CPU est ici appelé USER.

***A priori*, toute dérive importante du temps SYST et/ou du temps ELAPS doit questionner (cf. §6.2).**

Détaillons les niveaux d'impressions de MESURE_TEMPS/NIVE_DETAIL=niv (**défaut=1)**:

/niv=0, Aucune impression relative au monitoring en fin de chaque commande,

/niv=1, Impressions synthétiques des trois types de temps pour les calculs élémentaires/assemblages et pour la résolution de systèmes linéaires associés :

```
#1.Resolution.des.systemes.lineaires........CPU.(USER+SYST/SYST/ELAPS):...7.52...0.79.. 11.22
#2.Calculs.elementaires.et.assemblages...   CPU.(USER+SYST/SYST/ELAPS):..15.07...0.70...15.77
```

Figure 4.1.2-a : Consommations en temps dans une commande Aster type STAT_NON_LINE ou CALC_CHAMP avec niv=1 en séquentiel (extrait d'un .mess).

/niv=2, Impressions détaillées des temps pour les deux grands classes de calculs : calculs élémentaires/assemblages et résolution de systèmes linéaires associés. Ce type d'information peut renseigner quant à l'impact d'une modification du fichier de commande ou des paramètres de lancement du calcul (mémoire, parallélisme...). Par exemple, sachant que le parallélisme MPI ne permet potentiellement de diminuer (cf. doc. U2 dédiée) que les étapes 1.3/1.4/2, **cela est peu utile de paralléliser un calcul**[8] sur des dizaines de processeurs si l'étape 1.2 prend 20% du temps total[9] !

```
#1.Resolution.des.systemes.lineaires......... CPU.(USER+SYST/SYST/ELAPS):...7.72...0.82...8.72
#1.1.Numerotation,.connectivité.de.la.matrice CPU.(USER+SYST/SYST/ELAPS):...0.21...0.02...0.31
#1.2.Factorisation.symbolique.................CPU.(USER+SYST/SYST/ELAPS):...0.58...0.05...1.28
#1.3.Factorisation.numerique.(ou.precond.)....CPU.(USER+SYST/SYST/ELAPS):...6.78...0.73...7.71
#1.4.Resolution...............................CPU.(USER+SYST/SYST/ELAPS):...0.15...0.02...0.35
#2.Calculs.elementaires.et.assemblages........CPU.(USER+SYST/SYST/ELAPS):..28.87...0.64...29.47
#2.1.Routine.calcul...........................CPU.(USER+SYST/SYST/ELAPS):..26.61...0.56...26.61
#2.1.1.Routines.te00ij........................CPU.(USER+SYST/SYST/ELAPS):..24.58...0.07...25.78
#2.2.Assemblages..............................CPU.(USER+SYST/SYST/ELAPS):...2.26...0.08...3.36
#2.2.1.Assemblage.matrices....................CPU.(USER+SYST/SYST/ELAPS):...2.02...0.06...3.12
#2.2.2.Assemblage.seconds.membres.............CPU.(USER+SYST/SYST/ELAPS):...0.24...0.02...0.37
```

Figure 4.1.2-b : Consommations en temps dans une commande Aster type STAT_NON_LINE ou CALC_CHAMP avec niv=2 en séquentiel (extrait d'un .mess).

/niv=3, Idem que niv=2 mais l'impression est faite pour chaque pas de temps ou incrément de calcul.

En mode parallèle MPI (cf. doc. U2 sur le parallélisme ou doc. U4.50.01), on rajoute la valeur moyenne, sur tous les processeurs, des temps consommés ainsi que leur écart-type. Cette information est intéressante pour repérer d'**éventuels déséquilibres de charges**. Ils peuvent être dus à une distribution de données[10] non homogène en nombre de mailles, en terme de complexité de loi de comportement, en accès mémoire aux structures de données...

```
#1.Résolution.des.systèmes.linéaires..........CPU.(USER+SYST/SYST/ELAPS):...0.29...0.00...0.35
   ....(moyenne....diff..procs).................CPU.(USER+SYST/SYST/ELAPS):...0.30...0.00...0.47
....   (écart-type.diff..procs).......          CPU.(USER+SYST/SYST/ELAPS):...0.01...0.00...0.05
```

Figure 4.1.2-c : Consommations en temps dans une commande Aster type STAT_NON_LINE ou CALC_CHAMP avec niv=1 en mode parallèle MPI (extrait d'un .mess).

8 Cela va tout de même permettre de baisser les consommations en temps et la quantité de mémoire requise pour la construction et la résolution du système linéaire. Du coup, les autres fonctionnalités tel le contact-frottement pourraient s'en trouver (un peu) accélérées car elles disposeraient ainsi de plus de place en RAM.

9 Le gain en temps (speed-up) est borné à un facteur 5, même sur des centaines de processeurs !

10 Dans le flot de données distribuées de JEVEUX ou d'éventuels outils externes parallèles (MUMPS, PETSC).

D'autre part, il arrive que le gestionnaire mémoire d'*Aster* (JEVEUX) **décharge des objets sur disque** pour libérer de la RAM (cf. §5.1). Suivant la configuration matérielle, la charge machine et la taille du problème, ces déchargements sont susceptibles de ralentir de manière non négligeable le déroulement du calcul. **Lorsque des déchargements se sont produits, au cours d'un opérateur, on trace le cumul de leurs temps consommés** (USER+SYST, SYST, ELAPS) en fin d'opérateur. Cela permet par la suite d'affiner d'éventuels diagnostics (cf. §6.2).

```
#4 Dechargement de la memoire sur disque     CPU (USER+SYST/SYST/ELAPS):    0.04      0.04
0.04
```

Figure 4.1.2-d : Surcoûts en temps dûs aux déchargements sur disque de JEVEUX (extrait d'un .mess).

Remarque:

On récapitule le volume total récupéré et le nombre d'occurences de ce mécanisme en fin de fichier message (cf. §5 .1).

4.2 Temps consommés globaux

En fin de fichier message sont tracés systématiquement **la somme des consommations** CPU, CPU+SYST, SYST et le reliquat de temps non utilisé (entre le temps mentionné par l'utilisateur dans *Astk* et le temps CPU+SYST).

***A priori* , toute dérive importante du temps SYSTEME doit questionner (cf. § 6.2).**

```
<I> < INFORMATION TEMPS D'EXECUTION > (EN SECONDE)
      TEMPS CPU TOTAL ............... 2160.55
      TEMPS CPU USER TOTAL ......... 2099.33
      TEMPS CPU SYSTEME TOTAL ......   61.22
      TEMPS CPU RESTANT .............  439.45
```

Figure 4.2-a : Consommations globales en temps d'un calcul (extrait d'un .mess).

4.3 Cas particulier de DYNA/STAT_NON_LINE

Dans ces commandes de dynamique/statique non linéaires, on trace en standard (INFO=1) **pour chaque pas de temps ou incrément de calcul**, en supplément du tableau de décroissance des résidus (le niveau de détails est géré par le mot-clé AFFICHAGE des commandes) :

- Les champs stockés (sélectionnés par le mot-clé ARCHIVAGE),
- La ventilation des temps CPU consommés et, éventuellement, le nombre d'itérations associés (ex. processus de Newton),
- Des blocs d'affichages dédiés (ex. contact-discret).

L'**archivage des champs peut être coûteux en temps (surtout en parallèle) et en mémoire**. Il est donc être intéressant d'analyser la liste des champs archivés afin éventuellement de les limiter.

```
ARCHIVAGE DES CHAMPS:
   CHAMP STOCKE : CONT_NOEU        INSTANT :  5.00000E+02  NUMERO D'ORDRE :    50
   CHAMP STOCKE : DEPL             INSTANT :  5.00000E+02  NUMERO D'ORDRE :    50
   CHAMP STOCKE : SIEF_ELGA        INSTANT :  5.00000E+02  NUMERO D'ORDRE :    50
   CHAMP STOCKE : VARI_ELGA        INSTANT :  5.00000E+02  NUMERO D'ORDRE :    50
   CHAMP STOCKE : VITE             INSTANT :  5.00000E+02  NUMERO D'ORDRE :    50
   CHAMP STOCKE : ACCE             INSTANT :  5.00000E+02  NUMERO D'ORDRE :    50

TEMPS CPU CONSOMME DANS CE PAS DE TEMPS :  0 s
   TEMPS PAR ITERATION DE NEWTON  :  0 s                  - NBRE NEWT.:     2
   TEMPS ARCHIVAGE                :  0 s
   TEMPS CREATION NUMEROTATION    :  0 s                  - NBRE NUME.:     0
   TEMPS FACTORISATION MATRICE    :  0 s                  - NBRE FACT.:     0
   TEMPS INTEGRATION COMPORTEMENT :  0 s                  - NBRE INTE.:     3
   TEMPS RESOLUTION K.U = F       :  0 s                  - NBRE RESO.:     2
   TEMPS RESOLUTION CONTACT       :  0 s                  - NBRE ITER.:     2
   TEMPS AUTRES OPERATIONS        :  0 s

CONTACT DISCRET:
   NOMBRE D'ITERATIONS DE CONTACT        :        2
```

Champs archivés
A limiter (surtout en parallèle)

Ventilation temps CPU/itérations

Seules ces parties peuvent bénéficier du parallélisme

Bloc d'affichages dédié (contact...)

```
   NOMBRE D'ITERATIONS DE REAC. GEOM     :        2
   NOMBRE FINAL DE LIAISONS DE CONTACT   :        0
   TEMPS TOTAL APPARIEMENT               :  0 s
   TEMPS TOTAL RESOLUTION                :  0 s
```

Figure 4.3-a : Affichage à chaque pas de temps de DYNA/STAT_NON_LINE avec INFO=1 en mode séquentiel (extrait d'un .mess).

En fin d'opérateur sont résumées les statistiques globales sur tout le transitoire. Ces temps CPU se retrouvent dans la statistique globale de fin d'opérateur mentionnée au paragraphe précédent. Par contre, leur granularité est plus fine et adaptée aux différentes étapes de l'opérateur.

```
-------------------------------------------------------------------------------------------
STATISTIQUES SUR LE TRANSITOIRE
-------------------------------------------------------------------------------------------
   NOMBRE DE PAS DE TEMPS                  :    100
   NOMBRE D'ITERATIONS DE NEWTON           :    200
   NOMBRE D'ITERATIONS DE CONTACT (ALGO)   :    456
   NOMBRE D'ITERATIONS DE CONTACT (GEOM)   :    200
   NOMBRE DE CREATION DE NUMEROTATION      :      1
   NOMBRE DE FACTORISATION DE MATRICE      :      2
   NOMBRE D'INTEGRATION DE COMPORTEMENT    :    201
   NOMBRE DE RESOLUTION K.U = F            :    200
   TEMPS POUR CREATION NUMEROTATION        :  10 s
   TEMPS POUR FACTORISATION MATRICE        : 100 s
   TEMPS POUR INTEGRATION COMPORTEMENT     :  3 m  8 s
   TEMPS POUR RESOLUTION K.U = F           :  9 s
   TEMPS POUR CONTACT (APPARIEMENT)        : 17 s
   TEMPS POUR CONTACT (ALGORITHME)         :  2 m 30 s
-------------------------------------------------------------------------------------------
```

Figure 4.3-b : Statistiques globales à l'issu de DYNA/STAT_NON_LINE avec INFO=1 (extrait d'un .mess).

5 Consommation mémoire RAM

A chaque instant, l**a mémoire RAM utilisée par *Code_Aster*** peut être le cumul de trois composantes.

- La **mémoire allouée directement par les sources du code.** Il s'agit principalement de celle allouée par **le progiciel JEVEUX**[D6.02.01] pour gérer toutes les structures de données F77 du code. Ce progiciel permet aussi de décharger sur disque ces objets (fonctionnalité Out-Of-Core). Suivant la taille du problème, celle de la RAM et les préconisations du programmeur, JEVEUX effectue des échanges de données plus ou moins prononcés entre le disque et la RAM.
- La **mémoire utilisée par un logiciel externe** sollicité par le calcul *Aster* (MUMPS/PETSc, METIS/SCOTCH, HOMARD, MISS3D...). La plupart des logiciels externes fonctionnent en In-Core. C'est-à-dire qu'ils ne gèrent pas directement les éventuels débordements mémoire et qu'ils sous-traitent cette tâche au système d'exploitation. D'autres, comme le solveur linéaire MUMPS, sont potentiellement OOC et ils gèrent eux-même explicitement le déchargement sur disque de certains gros objets (cf. mot-clé SOLVEUR/GESTION_MEMOIRE[U4.50.01]).
- La **mémoire requise par le système** (chargement d'une partie de l'exécutable, couche réseau en parallèle...) et par le superviseur et les librairies Python.

Lorsqu'on utilise un opérateur *Code_Aster* requérant la construction et la résolution de systèmes linéaires (par ex. STAT_NON_LINE ou CALC_MODES), **les limitations mémoires sont souvent imposées par le solveur linéaire** (cf. [U4.50.01][U2.08.03]). Lorsqu'il s'agit de solveurs internes (LDLT, MULT_FRONT, GCPC), leurs consommations RAM se retrouvent dans les affichages JEVEUX. Par contre, lorsqu'on utilise un produit externe (MUMPS ou PETSC), il faut alors tenir compte de sa consommation propre (qui se substituent en grande partie à celle de JEVEUX).

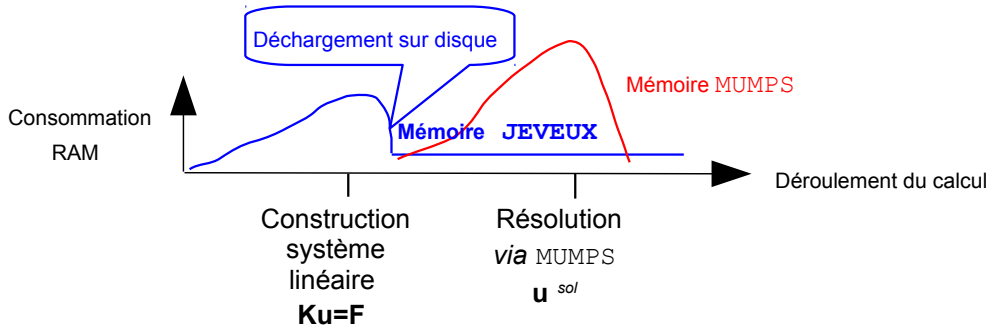

Figure 5-a : Schéma d'évolution de la consommation RAM au cours d'un calcul standard en mécanique linéaire.

5.1 Calibration mémoire d'un calcul

Il est souvent intéressant de calibrer au niveau mémoire RAM un calcul *Aster.* Cela peut, par exemple, permettre d'optimiser son placement dans une classe batch (séquentielle ou parallèle) ou tout simplement éviter son arrêt brutal du fait d'un défaut de mémoire. Pour ce faire, on peut procéder comme suit:

- **Etape n°1** : repérer dans le déroulement du calcul l'opérateur qui semble le plus dimensionnant en taille de problème (c'est souvent le STAT/DYNA_NON_LINE ou le CALC_MODES ... traitant le plus gros modèle).
- **Etape n°2:** si ce n'est pas le cas, paramétrer le bloc SOLVEUR de cet opérateur avec METHODE='MUMPS' et GESTION_MEMOIRE='EVAL'. A l'issue de la calibration, éventuellement penser à remettre l'ancien paramétrage.
- **Etape n°3:** lancer le calcul tel quel avec des paramètres mémoire et temps modestes par rapport aux consommations usuelles de ce type de calcul. En effet, avec cette option,

Code_Aster va juste construire le premier système linéaire de l'opérateur et le transmettre à MUMPS pour analyse. Ce dernier ne va pas effectuer son étape la coûteuse (en temps et surtout en mémoire) de factorisation numérique. Une fois cette analyse achevée, les estimés mémoire de MUMPS (MUE_{IC} et MUE_{OOC}), jointes à celle de JEVEUX (JE_{OOC}), sont tracées dans le fichier message (cf. figure 5.1-a). Puis le calcul s'arrête en ERREUR FATALE afin de permettre de passer le plus rapidement possible à l'étape suivante.

```
******************************************************************************
 - Taille du système linéaire: 500000

 - Mémoire RAM minimale consommée par Code_Aster                    : 200 Mo
 - Estimation de la mémoire Mumps avec GESTION_MEMOIRE='IN_CORE'    :3500 Mo
 - Estimation de la mémoire Mumps avec GESTION_MEMOIRE='OUT_OF_CORE' : 500 Mo
 - Estimation de l'espace disque pour Mumps avec GESTION_MEMOIRE='OUT_OF_CORE':2000 Mo

  ===> Pour ce calcul, il faut donc une quantité de mémoire RAM au minimum de
        - 3500 Mo si GESTION_MEMOIRE='IN_CORE',
        -  500 Mo si GESTION_MEMOIRE='OUT_OF_CORE'.
 En cas de doute, utilisez GESTION_MEMOIRE='AUTO'.
******************************************************************************
```

JE_OOC, MUE_IC, MUE_OOC

Figure 5.1-a : Affichage dans le fichier message en mode 'EVAL.

- **Etape n°4**: exploitation des estimés mémoires proprement dites.

Pour relancer le calcul avec le solveur linéaire MUMPS, on a directement accès aux valeurs « plancher » de la mémoire RAM indispensables au calcul. On distingue deux cas de figure suivant le mode de gestion mémoire de MUMPS choisit: In-Core (valeur $max(JE_{OOC}, MUE_{IC})$ si GESTION_MEMOIRE= ' IN_CORE ') ou Out-Of-Core (valeur $max(JE_{OOC}, MUE_{OOC})$ si GESTION_MEMOIRE= ' OUT_OF_CORE '). Suivant la machine/les classes batch dont on dispose, il faut donc relancer le calcul complet en modifiant aussi éventuellement ce paramètre du bloc SOLVEUR .
Ces estimées sont établies pour une configuration informatique et numérique donnée : plate-forme matérielle, librairies, nombre de processus MPI, renumeroteur, pré-traitements...

- **Etape n°4bis**: par contre, si on souhaite relancer le calcul en changeant de solveur linéaire ou un des paramètres numériques, il est plus difficile d'en déduire l'estimée mémoire adaptée. La combinatoire et la variabilité des possibilités sont trop importantes. On peut toutefois proposer quelques règles empiriques (en mode séquentiel). Grossièrement, si au lieu de METHODE='MUMPS' on choisit 'MULT_FRONT' l'estimée mémoire devrait rester licite (avec RENUM='METIS' valeur par défaut). Avec 'LDLT', ce chiffre doit être significativement revu à la hausse. Avec 'PETSC'/'GCPC'+'LDLT_SP', il peut être sans doute réduit à $max(JE_{OOC}, MUE_{IC/OOC}/2)$. Avec 'PETSC'/'GCPC'+'LDLT_INC' il peut être sans doute réduit à un facteur fois JE_{OOC} suivant le niveau de remplissage du préconditionneur (mot-clé NIVE_REMPLISAGE cf.[R6.01.02]).

- **Etape n°5:** relancer le calcul complet en modifiant éventuellement le paramétrage du bloc SOLVEUR et en paramétrant Astk avec la valeur déduite à l'étape n°4 (menu «Mémoire totale (Mo)» de la figure 5.2-b).

Remarque:

Cette procédure de calibration mémoire d'une étude n'est disponible que depuis la restitution du mot-clé SOLVEUR/GESTION_MEMOIRE (à partir de la version v11.2 de Code_Aster). Pour des versions plus anciennes, si le calcul a déjà été lancé (et si on dispose d'un fichier de message exploitable), on peut, a minima, reprendre prudemment comme estimation optimale de la mémoire requise, la plus grande valeur sur tous les processeurs du Vmpeak du dernier opérateur. Sinon, il faut procéder comme mentionné dans les versions antérieures de ce document:

- ***Etape n°1:** comme ci-dessus.*

- **Etape n°2:** paramétrer le bloc SOLVEUR avec METHODE='MUMPS', 'OUT_OF_CORE'='OUI' et INFO=2.
- **Etape n°3:** lancer le calcul en limitant la consommation en temps (par exemple, avec peu de pas de temps ou peu de recherche de modes propres).
- **Etape n°4:** suivant le mode de gestion mémoire de MUMPS requis (IC ou OOC), prendre le maximum de l'estimation MUMPS et de la consommation JEVEUX.

5.2 Consommations mémoire JEVEUX

Des **éléments pour chiffrer les consommations mémoire de JEVEUX** sont tracés en fin de chaque opérateur *Aster* (cf. figure 5.2-a). A grands traits, on peut les expliciter de la façon suivante:

- JE_{OOC} fournit la **taille minimale** ('Minimum') **dont à besoin JEVEUX pour fonctionner jusqu'à cet opérateur**. Il sera alors complètement Out-Of-Core (OOC). En deçà de cette valeur, le calcul n'est pas possible (dans la configuration retenue).
- JE_{IC} fournit une borne inférieure ('Optimum'), jusqu'à cet opérateur, de la mémoire JEVEUX nécessaire pour fonctionner complètement en In-Core (IC): JE'_{IC}. Avec au moins cette valeur de mémoire RAM paramétrée dans Astk (cf. MEM_{ASTK}), le calcul se déroulera de manière optimale: il n'y a pas de risque de planton dû à un manque de mémoire et les accès aux structures de données purement Aster sont peu ralentis par les déchargements sur disque.

Ces deux chiffres sont forcément inférieurs à la valeur totale de mémoire RAM consacrée au calcul (paramétrée dans *Astk*), notée MEM_{ASTK}.


```
# Statistiques mémoire(Mo): 15521.78/ 8920.14/ 1603.77/ 248.27(VmPeak/VmSize/Optimum/Minimum)
```

Figure 5.2-a : Statistiques globales à l'issue de chaque commande (extrait d'un .mess).

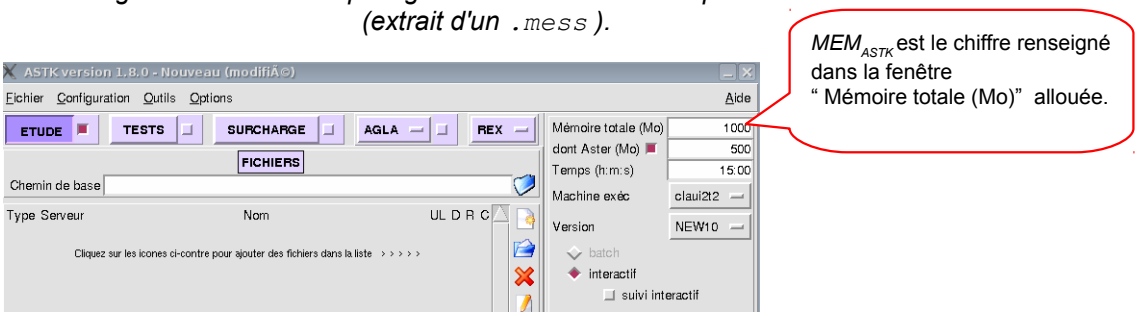

Figure 5.2-b : Paramétrage dans Astk de la mémoire RAM.

Remarque:

Si il n'y a eu aucun mécanisme global de libération (cf. paragraphe ci-dessous), JE_{IC} représente véritablement la mémoire JEVEUX requise pour fonctionner en IC. Sinon, cette dernière estimation doit être proche de la somme[11] *$JE'_{IC} = JE_{IC} + .$ «le gain moyen procuré par chaque libération». Il ne sert donc à rien (si on n'utilise pas de produit externe) de paramétrer le calcul avec une valeur très supérieure à JE'_{IC}.*

11 Une stratégie pour déterminer exactement le point de fonctionnement IC pour JEVEUX est d'augmenter progressivement la valeur de MEM_{ASTK} La valeur de JE_{IC} doit alors augmenter. Dès qu'elle ne bouge plus, c'est qu'on a atteint le point de fonctionnement optimal permettant à JEVEUX de fonctionner totalement en IC: JE'_{IC}.

Lors du paramétrage du calcul, il faut donc établir un compromis entre sa rapidité et ses consommations mémoire. En hiérarchisant les consommations mémoires de toutes les commandes, on peut repérer celle qui dimensionne le calcul. A configuration de calcul fixée (nombre de processeurs), plus l'espace mémoire réservé à JEVEUX sera grand, plus le calcul sera In-Core (IC) et donc plus il sera rapide. On peut aussi, suivant les contingences des files d'attente batch, découper son calcul en différentes POURSUITE de manière à panacher et ainsi optimiser les paramétrages «temps calcul/mémoire».

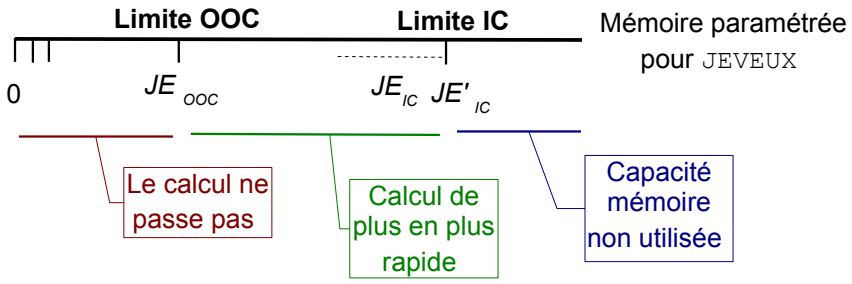

Figure 5.2-c :Signification des affichages liés à JEVEUX dans le fichier message.

En cours de calcul, **JEVEUX peut décharger une grande partie des objets de la RAM sur le disque**. Ce mécanisme est soit:

- Automatique (il n'y a plus assez d'espace RAM pour créer un objet ou le rapatrier),
- Géré par le programmeur (par ex. appel à ce mécanisme de libération juste avant de céder la main à MUMPS).

Les statistiques concernant ce mécanisme sont récapitulées en fin de fichier message (cf. §5.2-d) pour l'ensemble des commandes. Évidemment, plus ce mécanisme intervient (et sur de gros volumes de mémoire libérés), plus le calcul s'en trouve ralenti (temps système et elapsed qui grandissent et temps CPU stable). Ce phénomène peut se trouver accentué en mode parallèle (contentions mémoires simultanées) suivant la répartition des processeurs sur les nœuds de calcul et suivant les caractéristiques de la machine.

```
STATISTIQUES CONCERNANT L'ALLOCATION DYNAMIQUE :
     TAILLE CUMULEE MAXIMUM            :                1604  Mo.
     TAILLE CUMULEE LIBEREE            :               52117  Mo.
     NOMBRE TOTAL D'ALLOCATIONS        :             1088126
     NOMBRE TOTAL DE LIBERATIONS       :             1088126
     APPELS AU MECANISME DE LIBERATION :                   1
     TAILLE MEMOIRE CUMULEE RECUPEREE  :                1226  Mo.
     VOLUME DES LECTURES               :                   0  Mo.
     VOLUME DES ECRITURES              :                1352  Mo.

   MEMOIRE JEVEUX MINIMALE REQUISE POUR L'EXECUTION :     248.27 Mo
     - IMPOSE DE NOMBREUX ACCES DISQUE
     - RALENTIT LA VITESSE D'EXECUTION
   MEMOIRE JEVEUX OPTIMALE REQUISE POUR L'EXECUTION :    1603.77 Mo
     - LIMITE LES ACCES DISQUE
     - AMELIORE LA VITESSE D'EXECUTION
```

Figure 5.2-d :Statistiques du mécanisme JEVEUX de libération (.mess).

Ces statistiques récapitulent aussi les estimées JE_{IC} et JE_{OOC} détaillées précédemment.

Remarque:

On trace les éventuels surcoûts en temps de ces déchargements à la fin des opérateurs concernés (cf. §4.1.1). Dans le cas d'un manque de RAM, on peut diagnostiquer celui-ci via le paramètre 'VmPeak' (cf. §5.4).

5.3 Consommations mémoire de produits externes (MUMPS)

Pour l'instant, le seul produit externe dont la **consommation mémoire RAM soit vraiment dimensionnante, est le produit MUMPS** . On peut tracer ses consommations (In-Core et Out-Of-Core) de deux manières:

- Par **un pré-calcul** très rapide et peu coûteux en mémoire *via* le mode GESTION_MEMOIRE='EVAL' (cf. § 5.1). Mais il ne s'agit que d'estimations (valeurs ME_{IC} et

ME_{OOC} de figure 5.1-a). Elles peuvent donc être un peu pessimistes (surtout en Out-Of-Core et/ou en parallèle). En mode parallèle, on ne trace que les estimées du processus MPI le plus gourmand.

• Par **le calcul standard** (limité si possible à quelques pas de temps ou à quelques modes propres) en rajoutant le mot-clé INFO=2 dans la commande supposée la plus coûteuse. On récapitule alors dans le fichier message (cf. 5.3-a), non seulement les estimées mémoire ME_{IC} et ME_{OOC} précédentes (colonnes 'ESTIM IN-CORE' et 'ESTIM OUT-OF-CORE'), mais aussi la consommation réelle du mode choisi $MR_{IC/OOC}$ (ici en mode OOC car affichage 'RESOL OUT-OF-CORE'). Par exemple, dans la figure 5.3-a, il faut lire: «MUMPS estime avoir besoin d'au moins $ME_{OOC}=478\text{Mo}$ par processeur pour fonctionner en OOC. Pour passer en IC (calcul plus rapide), il estime avoir besoin d'au moins $ME_{IC}=964\text{Mo}$; En pratique, il a consommé exactement $MR_{OOC}=478\text{Mo}$ en OOC (l'estimé est donc parfaite: $ME_{OOC}=MR_{OOC}$!).
En mode parallèle, on récapitule les chiffres par processus MPI, ainsi que leurs minima, maxima et moyennes. C'est un affichage à vocation d'expertise qui est plus détaillé que l'affichage du mode GESTION_MEMOIRE='EVAL'. Outre ces informations de consommation RAM, il récapitule aussi des éléments liés au type de problème, à sa difficulté numérique (erreurs) et à sa distribution parallèle (équilibre de charge).

```
*****************************************************************************
 <MONITORING DMUMPS 4.8.4 >
 TAILLE DU SYSTEME    210131
 RANG     NBRE MAILLES     NBRE TERMES K     LU FACTEURS
 N    0 :          5575          685783        45091154
 N    1 :          8310         1067329        51704129
 N    2 :         11207         1383976        53788943
                                    ...
 N    7 :          8273         1039085         4560659
 ---------------------------------------------------------
TOTAL   :         67200         8377971       256459966

  MEMOIRE RAM ESTIMEE ET REQUISE              PAR MUMPS EN MO(FAC_NUM + RESOL)
 RANG ASTER : ESTIM IN-CORE | ESTIM OUT-OF-CORE | RESOL. OUT-OF-CORE
 N    0 :          869 ,          478                478
 N    1 :          816            439                439
 N    2 :          866            437                437
                                    ...
 N    7 :          754            339                339
 ---------------------------------------------------------------------------
MOYENNE :       5.92D+02       3.50D+02       3.50D+02
 ---------------------------------------------------------------------------
MINIMUM :       7.54D+02       3.39D+02       3.39D+02
 ---------------------------------------------------------------------------
MAXIMUM :       9.64D+02       4.78D+02       4.78D+02
 ---------------------------------------------------------------------------
 *****************************************************************************
```

Figure 5.3-a :Consommations mémoires par processeur (estimations et réalisés) du produit externe MUMPS (affichage dans le .mess si INFO=2).

Remarques :

Suivant le type de calcul (séquentiel, parallèle centralisé ou distribué), le nombre de processeurs, le mode d'utilisation de JEVEUX (mot-clé SOLVEUR/MATR_DISTRIBUEE) et du gestionnaire mémoire MUMPS (mot-clé SOLVEUR/GESTION_MEMOIRE), la hiérarchie mémoire peut cependant être bousculée . En mode parallèle distribué, le pic mémoire JEVEUX va baisser avec le nombre de processeurs dès que MATR_DISTRIBUE va être activé. Cela va aussi être le cas pour MUMPS, quelque soit le mode de parallélisme (surtout en IC). D'ailleurs, le passage de MUMPS du mode IC au mode OOC va aussi drastiquement faire chuter son pic mémoire.
L'affichage en INFO=2 des consommations RAM de MUMPS par processeur renseigne quant à d'éventuels déséquilibrages de charge mémoire. On peut essayer de les limiter en modifiant l'heuristique d'ordonnancement du produit: paramètres SOLVEUR/PRETRAITEMENTS et RENUM ou le nombre de processeurs.

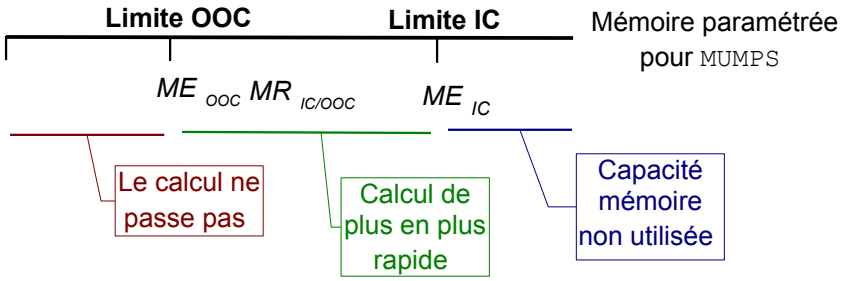

Figure 5.3-b :Signification des affichages liés à MUMPS dans le fichier message.

5.4 Consommations système

Les seules consommations systèmes mentionnées dans le fichier message concernent la mémoire globale du job à cet instant du calcul (VmSize) et son pic depuis le début (VmPeak). Elle est tracée à l'issue de chaque commande (cf. chiffres SYS_1/SYS_2 figure 5.2a). Le VmPeak est récapitulée en fin de fichier message (cf. figure 5.4a).

```
MAXIMUM de MEMOIRE UTILISEE PAR LE PROCESSUS          :    15107.89 Mo
   - COMPREND LA MEMOIRE CONSOMMEE PAR JEVEUX,
     LE SUPERVISEUR PYTHON, LES LIBRAIRIES EXTERNES
```

SYS 1

Figure 5.4-a : Vmpeak final du processus Unix.

A grands traits VmPeak renseigne quant au pic en taille mémoire causé par tous les exécutables actifs du job *Aster*. Il faut veiller à ce que ce chiffre reste inférieur à la mémoire physique dont dispose le job. Sinon, le système va, jusqu'à un certain point, « swapper » et cela va ralentir le calcul. Au niveau du progiciel JEVEUX cela se traduira par plus de déchargements (cf. §4.1.2 et §5.2) et, au niveau des produits externes, par un écart ELAPS/USER grandissant (tracé en fin de commande cf. §4.1.1).

•

Remarques si SOLVEUR='MUMPS':

- *Depuis la restitution du mot-clé SOLVEUR/GESTION_MEMOIRE (à partir de la version v11.2 de Code_Aster), on permet au solveur de « s'étaler » en mémoire, en complément des allocations JEVEUX /Python/librairies, jusqu'à occuper la taille maximale allouée. Cela permet au produit d'être plus rapide et moins sourcilleux sur ses besoins mémoire en terme de pivotage. Par contre, corolaire de cette stratégie d'occupation optimum des ressources mémoire, le VmPeak affiché dans le fichier message devient dépendant de la mémoire allouée au calcul (MEM ASTK ou borne de la classe batch). Il ne peut donc pas servir, comme auparavant, à calibrer les consommations mémoires minimales de l'étude. Pour ce faire, il faut plutôt appliquer la stratégie détaillée au § 5.1*

6 Quelques conseils pour optimiser les performances

On formule ici **quelques conseils pour aider l'utilisateur à tirer parti des diagnostics tracés** dans le fichier message. Mais il faut bien être conscient qu'il n'y a pas de recette universelle pour optimiser les performances globales d'un calcul. Cela dépend du type d'étude, des aspects logiciels et matériels de la machine, voire de sa charge !

Le paramétrage par défaut et les affichages/alarmes du code proposent un fonctionnement équilibré et instrumenté. Mais, pour être sûr d'avoir utilisé au mieux les capacités de sa machine, l'utilisateur doit rester attentif aux éléments décrits dans ce document ainsi qu'aux conseils qui fourmillent dans les documentations des commandes.

On liste ci-dessous et, de manière non exhaustive, plusieurs questions qu'il est intéressant de se poser lorsqu'on cherche à optimiser les performances de son calcul. Bien sûr, certaines questions (et réponses) sont cumulatives et peuvent donc s'appliquer simultanément.

6.1 Concernant les caractéristiques du problème

Au vu des éléments du §3, on peut formuler deux règles empiriques:

- Plus la **taille du problème** (N) et/ou **le remplissage de la matrice** (NNZ) augmentent, plus la construction, et surtout, la résolution du système linéaire vont être coûteux (CPU/RAM).
- L'augmentation de la **proportions de Lagrange** (N_L/N) peut rendre plus difficile la résolution du système linéaire (temps d'exécution, qualité de la solution).
- La taille du problème **dimensionne le nombre maximum de processeurs** qu'il est pertinent de consacrer à son calcul parallèle: une granularité d'au moins 20.10^3 degrés de liberté par process MPI est requise.

6.2 Concernant les temps consommés

Pour diminuer les temps CPU, l'utilisateur *Aster* dispose de différents outils :

- Si **l'essentiel des coûts concerne les calculs élémentaires/assemblages et/ou les résolutions de systèmes linéaires** (cf. §4.1) on conseille d'utiliser *Aster* en mode parallèle. Il est alors préférable d'utiliser le solveur linéaire MUMPS en mode MPI distribuée ou le solveur MULT_FRONT en OpenMP. La première stratégie permet aussi de faire baisser les consommations RAM par processeur.
- Si on **utilise déjà le solveur linéaire MUMPS,** on peut désactiver ses fonctionnalités d'OOC[12] (GESTION_MEMOIRE='IN_CORE') et d'amélioration de la qualité de la solution (RESI_RELA=-1.d0). Si la matrice est bien conditionnée et/ou non symétrique, on peut aussi essayer des paramètres de relaxation du solveur linéaire (FILTRAGE_MATRICE, MIXER_PRECISION, SYME).
- Si on effectue **un calcul non linéaire** on peut tester différents paramètres de relaxation du solveur non linéaire (REAC_INCR, REAC_ITER, SYME).
- Si on effectue un **calcul modal**, on conseille l'utilisation de la méthode IRAM (METHODE='SORENSEN') et le découpage du spectre recherché en plusieurs bandes

12 Cela peut être intéressant lorsqu'on constate des surcoûts d'I/O dans les étapes 1.3/1.4 *via* un temps SYST non négligeable et un temps ELAPS très supérieur au temps USER.

fréquentielles (*via* l'opérateur CALC_MODES avec OPTION='BANDE' découpée en plusieurs sous-bandes).

- De manière générale, plus **le mode de fonctionnement de JEVEUX (et MUMPS) est en IC** (cf. §5.1), plus le calcul est rapide. Ces gains ne sont cependant pas très importants comparés à ceux que procurent le parallélisme et le choix d'un solveur linéaire (avec le paramétrage) adapté.

Pour chaque étape du calcul, on **doit normalement avoir un temps système faible** (SYST) et un cumul «temps CPU+temps système» (CPU+SYST) très proche du temps d'attente réel (ELAPS). Si ce n'est pas le cas, deux cas de figures classiques peuvent se présenter:

- **Le temps USER+SYS est très supérieur au temps ELAPS**. Le calcul utilise certainement le parallélisme OpenMP des stratégies parallèles 1c ou 2a décrites dans la doc. U2 sur le parallélisme et au §2.2 de la doc. U4.50.01. Cette situation n'est pas inquiétante (et même voulue), le principale étant que le temps de retour du calcul soit le plus bas possible !

- **Le temps ELAPS est très supérieur au temps CPU et/ou le temps SYS est important.** Le calcul pâtit probablement de contentions mémoires (swap système, I/O RAM/disque...).
  ✔ **Piste n°1:** Ce surcoût peut provenir des **déchargements globaux de JEVEUX**. Pour s'en convaincre il suffit d'aller lire, en fin de .mess, les statistiques concernant l'allocation dynamique (cf. §5.2) ou les consommations en temps des différents déchargements (cf. §4.1). Plus il y a eu d'appels aux mécanismes de libération et/ou de gros objets libérés, plus les temps SYST et ELAPS vont se dégrader. **Une solution est alors d'augmenter la taille mémoire dévolue à JEVEUX**.

  ✔ **Piste n°2:** Corollaire du constat précédent, le mode parallèle MPI a tendance a décupler les surcoûts dûs aux déchargements. En effet, la distribution des données induites par le parallélisme va diminuer la taille des objets JEVEUX (si SOLVEUR/MATR_DISTRIBUEE est activée) et donc limiter l'impact des déchargements. *A contrario*, ceux-ci risquent de s'effectuer en même temps et sur des processeurs contiguës. **Une solution palliative** peut alors consister à «gâcher» du processeur, **en entrelaçant les processeurs MPI actifs de processeurs dormants** (par ex. valeur ncpus d'*Astk* initialisée à 2).

  ✔ **Piste n°3*:*** Si on utilise le solveur linéaire MUMPS en OOC, le problème peut provenir d'un **grand nombres de descente-remontées** du produit (cf. étape 1.4 de §4.1). On peut les limiter en débranchant l'option de raffinement automatique (RESI_RELA=-1.d0) ou en repassant en mode IC (si les ressources mémoires le permettent).

6.3 Concernant la mémoire RAM consommée

- Si les consommations JE_{IC} et JE_{OOC} sont proches (quelques dizaines de pourcents), c'est que JEVEUX a dû souvent fonctionner en mode OOC. Il y a eu probablement de nombreux déchargements mémoire (cf. §4.1/5.2). Le temps de calcul peut en pâtir (surtout en parallèle). Il faut essayer de rajouter de la mémoire ou augmenter le nombre de processeur (si l'option MATR_DISTRIBUEE est activée).

- Le **calcul est souvent dimensionné en mémoire**, par le maximum, sur l'opérateur le plus gourmand, entre la valeur plancher de JEVEUX (JE_{OOC}) et la valeur utilisée par le solveur linéaire MUMPS (si on l'utilise). Pour diminuer ce chiffre on peut jouer sur plusieurs facteurs:
  ✔ **Si on utilise MUMPS et que ce dernier est prédominant** (c'est souvent le cas): utiliser plus de processeurs en MPI (paramètres mpi_nbcpu/mpi_nbnoeud d'*Astk*), passer en OOC (mot-clé SOLVEUR/GESTION_MEMOIRE). Si la matrice est bien conditionnée et/ou non symétrique, on peut aussi essayer des paramètres de relaxation du solveur linéaire (FILTRAGE_MATRICE, MIXER_PRECISION, SYME).

- ✔ **Si on utilise MUMPS et que ce dernier n'est pas prédominant:** utiliser la distribution d'objets JEVEUX de l'option MATR_DISTRIBUEE en mode parallèle.
- ✔ Si on utilise un **autre solveur linéaire**: utiliser MUMPS en parallèle voire GCPC/PETSC (en séquentiel/parallèle).

6.4 Concernant le parallélisme

- Si **l'essentiel des coûts concerne les calculs élémentaires/assemblages et/ou les résolutions de systèmes linéaires** (cf. §4.1) on conseille d'utiliser *Aster* en mode parallèle. Il est alors préférable d'utiliser le solveur linéaire MUMPS en mode MPI distribuée (cf. doc. U2 sur le parallélisme ou U4.50.01) ou le solveur MULT_FRONT en OpenMP.

- Il est préférable de limiter le parallélisme du calcul uniquement aux quelques opérateurs coûteux (en temps/mémoire) : STAT/DYNA_NON_LINE, CALC_MODES ... Et donc, si possible, de découper celui-ci en une succession de pré/post-traitements séquentiels et de calculs parallèles. Lors de longs calculs (quelques jours), cette stratégie permet en outre de mieux se prémunir contre d'éventuels arrêts en erreur en sauvegardant la base associée à chaque grande étape du calcul.

- Il est intéressant de **valider**, au préalable, **son calcul parallèle** en comparant quelques itérations en mode séquentiel et en mode parallèle. En outre, cette démarche permet aussi de **calibrer les gains maximums atteignables** (speed-up théoriques) et donc d'éviter de trop «gaspiller de processeurs». Ainsi, si on note f la portion parallèle du code (déterminée par exemple *via* run séquentiel préalable), alors le speed-up théorique $\mathbf{S}_p$ maximal accessible sur p processeurs se calcule suivant la formule d'Amdhal (cf. [R6.01.03] §2.4):

$$\mathbf{S}_p = \frac{1}{1-f+f/p}.$$

  Par exemple, si on utilise le parallélisme distribué par défaut (scénarios 1b+2b, cf. doc. U2 sur le parallélisme) et que les étapes 1.3/1.4 et 2. (cf. §4.1) représentent 90% du temps séquentiel ($f=0.90$), le speed-up théorique est borné à la valeur $\mathbf{S}_\infty = \frac{1}{1-0.9+0.9/\infty} = 10$! Et ce, quelque soit le nombre de process MPI alloués.

- Pour optimiser son calcul parallèle, il faut surveiller les éventuels **déséquilibres de charges** du flot de données (CPU et mémoire) et limiter les surcoûts dûs **aux déchargements mémoire** (JEVEUX et MUMPS OOC) (cf. §4.1)et aux **archivages de champs** (cf. §4.3). Pour gagner en temps calcul, il faut aussi proscrire toute procédure de retassage de la mémoire (commande FIN mot-clé RETASSAGE) qui est contre-productive en parallèle.

- Le **recours au parallélisme MPI** de MUMPS permet de gagner en temps CPU (sur les étapes parallélisées) et en mémoire RAM: grâce à la distribution de données de JEVEUX (si l'option MATR_DISTRIBUEE est activée) et, surtout, grâce à celle des objets MUMPS.

- **Quelques chiffres empiriques:** on conseille d'allouer au moins 20 000 degrés de liberté par processus MPI ; Un calcul thermo-mécanique standard bénéficie généralement, sur 32 processeurs, d'un gain de l'ordre de la dizaine en temps elapsed et d'un facteur 4 en mémoire RAM.